# ECEC 204: Design with Microcontrollers
# Dynamic LED Toggling

Instructor: Prof. Naga Kandasamy
ECE Department, Drexel University

In this assignment, you will write a C program that allows the user to dynamically control the blinking rate of the LED on the MSP432 microcontroller board. The solution to this assignment is due via BBLearn. Please submit original work. You can find detailed submission instructions towards the end of this document.

Let's prepare for this assignment by first discussing two related concepts: how to toggle the LED at a fixed rate; and how to accept user input from a Universal Asynchronous Receiver/Transmitter (UART) module on the microcontroller board.

## Toggling an LED on and off

Download and unzip the `blinking_led` project from the course website, and import the project into your workspace. The main function implements a simple, software-controlled blinking LED that is toggled on and off at some rate.

```
int main (void)
{
    volatile uint32_t i;

    WDT_A_hold (WDT_A_BASE); /* Stop watchdog timer */
    /* Set Pin 0 in Port 1, that is P1.0, to output direction */
    GPIO_setAsOutputPin (GPIO_PORT_P1, GPIO_PIN0);
    /* Set digital clock oscillator (DCO) to 12MHz. */
    CS_setDCOCenteredFrequency (CS_DCO_FREQUENCY_12);
    while (1) { /* Main loop that runs on the board */
        /* Toggle P1.0 output */
        GPIO_toggleOutputOnPin (GPIO_PORT_P1, GPIO_PIN0);
        for (i = 100000; i > 0; i --); /* Delay using nops */
    }
}
```

Connect the LaunchPad to the Computer; build and flash the application. Execute it and observe the functionality of the program. The red LED is connected to one of the general purpose input/output

(GPIO) pins available on the board, specifically to P1.0 (I/O port P1, pin 0) with a copper trace. External connections are unnecessary to control this LED. The GPIO pin P1.0 is configured to be an output pin in Line 7. Line 12 toggles the pin on and off, and Line 13 is an empty loop that implements the toggling delay within the `while` loop. The digitally controlled oscillator (DCO) is a clock source that drives the ARM core, and can be configured to generate frequencies anywhere between 1 to 48 MHz. In the example, the core is clocked at 12 MHz. Adjust the loop bound appropriately such that the LED toggles at approximately 1 Hz.

## Serial Communication via the UART

The goal is to learn how to use an UART (Universal Asynchronous Receiver Transmitter) to establish serial communication between the host PC and the microcontroller board. The UART takes bytes of data and transmits the individual bits in a sequential fashion. At the destination, a second UART re-assembles the bits into complete bytes. Various I/O functions are provided within the `uart_functions.c` file in the `uart_functions` project folder. The first step is to configure and enable the UART module on the board, as shown by the following code snippet.

```
1  const eUSCI_UART_Config uartConfig = {
2      EUSCI_A_UART_CLOCKSOURCE_SMCLK,          // SMCLK Clock Source
3      78,                                      // BRDIV = 78
4      2,                                       // UCxBRF = 2
5      0,                                       // UCxBRS = 0
6      EUSCI_A_UART_NO_PARITY,                  // No Parity
7      EUSCI_A_UART_LSB_FIRST,                  // LSB First
8      EUSCI_A_UART_ONE_STOP_BIT,               // One stop bit
9      EUSCI_A_UART_MODE,                       // UART mode
10     EUSCI_A_UART_OVERSAMPLING_BAUDRATE_GENERATION  // Oversampling
11 };
12
13 void initUART (void)
14 {
15     /* Select P1.2 and P1.3 in UART mode. */
16     MAP_GPIO_setAsPeripheralModuleFunctionInputPin (GPIO_PORT_P1,
17             GPIO_PIN2 | GPIO_PIN3, GPIO_PRIMARY_MODULE_FUNCTION);
18     /* Set digital clock oscillator (DCO) to 12MHz. */
19     CS_setDCOCenteredFrequency (CS_DCO_FREQUENCY_12);
20     /* Configure and enable UART. */
21     MAP_UART_initModule (EUSCI_A0_BASE, &uartConfig);
22     MAP_UART_enableModule (EUSCI_A0_BASE);
23     /* Enable Interrupts from UART to processor. */
24     MAP_UART_enableInterrupt (EUSCI_A0_BASE,
25                               EUSCI_A_UART_RECEIVE_INTERRUPT);
26     MAP_Interrupt_enableInterrupt (INT_EUSCIA0);
27     MAP_Interrupt_enableMaster ();
28     return;
29 }
```

We set the baud rate on the UART module to 9,600 bits per second (bps), and will match this baud

rate on the terminal running on the host PC. [1] The board uses P1.2 as the receive (`rx`) pin and P1.3 as the transmit (`tx`) pin. These pins are selected to be the I/O pins for the UART module in Line 17. The clock system module is configured to operate at a center frequency of 12 MHz in Line 20, which is needed to operate the UART module at a baud rate of 9,600.

Let us now turn our attention to setting up the terminal on the host PC to establish communication with the board via a serial port, also referred to as a COM port; for example COM3 or COM4. CCS has an internal terminal program to communicate with the MSP432 LaunchPad via the UART. The terminal window can be accessed by clicking on `View > Terminal`. The terminal application must then be configured appropriately to establish serial communication with the board (see Figures 2.11 and 2.12 in the textbook). Select `Serial` under terminal type and set the baud rate to 9,600. You can leave the data size, parity, and stop bits values unchanged. You must now find the port used by the MSP432 LaunchPad. Under Windows 10, open *Device Manager* and click on the item *ports (COM) & LPT* in the devices list. The port used by the MSP432 should be named *XDS110 Class Application/User UART*. On my computer, the MSP432 is connected to COM4. Once the program is flashed on to the board and executed, data received from the board is displayed in the terminal window. To transmit data to the board, the user can click anywhere within the window and type characters from the keyboard.

Instead of the built-in terminal application in CCS, we can also use an external terminal emulator such as PuTTY on the host PC to communicate with the board.[2] Be sure to operate PuTTY in serial mode with a baud rate setting of 9,600 bps.

[1] The baud rate is the rate at which information is transferred in a communication channel. In the serial port context, "9,600 baud" means that the serial port is capable of transferring a maximum of 9,600 bps.

[2] PuTTY can be freely downloaded from the following website: `www.chiark.greenend.org.uk/~sgtatham/putty/download.html`.

Let us return to our program running on the MSP432. Rather than *poll* the UART module constantly to check if we have received any data from the host—which wastes processor cycles—we allow the UART to interrupt the execution of the processor whenever data is available in its buffer.[3] When an interrupt is received by the processor, control is transfered to the following *interrupt service routine*.

```
1  void EUSCIA0_IRQHandler (void)
2  {
3      char c;
4      uint32_t status;
5
6      /* Obtain the current interrupt status of the UART. */
7      status = MAP_UART_getEnabledInterruptStatus (EUSCI_A0_BASE);
8
9      /* Clear UART interrupt sources. */
10     MAP_UART_clearInterruptFlag (EUSCI_A0_BASE, status);
11
12     /* Check if we received anything via the UART. */
13     if(status & EUSCI_A_UART_RECEIVE_INTERRUPT_FLAG) {
14         /* Read the data from the UART module. */
15         c = MAP_UART_receiveData (EUSCI_A0_BASE);
16
17         /* Echo it back to the host via the UART. */
18         MAP_UART_transmitData (EUSCI_A0_BASE, c);
19
20         /* Store the character in the ring buffer. */
21         ringBuffer.buffer[ringBuffer.in++] = c;
22         if (ringBuffer.in == BUFFER_SIZE)
23             ringBuffer.in = 0;
24     }
25
26     return;
27 }
```

Data available in the UART's receive buffer is read into a variable (Line 15) and stored in a *circular* or *ring buffer*, for our `receiveString` function to pick up. This function, shown below, reads the characters received by the UART one character at a time until carriage return (CR) is received and constructs the resulting string.

[3] Polling refers to the process of actively sampling the status of an external device by a program as a synchronous activity.

```
void readString (char *buffer)
{
    char c;
    char *ptr = buffer;

    for (;;) {
        if (ringBuffer.out == ringBuffer.in) /* Empty ring buffer. */
            continue;
        else {
            c = ringBuffer.buffer[ringBuffer.out++];
            if (ringBuffer.out == BUFFER_SIZE)
                ringBuffer.out = 0;

            if (c == 0x0D) { /* Check for carriage return. */
                *ptr = '\0'; /* Terminate the string. */
                break;
            }
            else {
                *ptr++ = c;
            }
        }
    }

    return;
}
```

The `writeString` function writes the provided string to the UART one character at a time to the until the end of string is reached.

```
void writeString (char *buffer)
{
    MAP_UART_transmitData (EUSCI_A0_BASE, '\n'); /* New line */
    MAP_UART_transmitData (EUSCI_A0_BASE, '\r'); /* Carriage return */

    char *ptr = buffer;
    while (*ptr != '\0') {
        MAP_UART_transmitData (EUSCI_A0_BASE, *ptr);
        ptr++;
    }

    MAP_UART_transmitData (EUSCI_A0_BASE, '\n');
    MAP_UART_transmitData (EUSCI_A0_BASE, '\r');
    return;
}
```

We must explicitly transmit the newline and carriage return characters, `\n` and `\r`, respectively, to the terminal to achieve proper formatting of the output.

To display an integer value on the terminal, the value needs to be first written to a string using the

C library function `sprintf` as shown in the following code example. This is because all data is transferred over the UART in terms of ASCII characters.[4] Conversely to read an integer or FP value from the terminal, the received string must be converted to the appropriate data type using the `atoi` or `atof` functions that are available in the standard C library.

```
/* Write an integer value to the UART. */
void writeInt (int value)
{
    char buf[BUFFER_SIZE];
    sprintf (buf, "%d", value); /* Convert integer to ASCII. */
    writeString (buf);
    return;
}

/* Read an integer from the terminal via the UART. */
void readInt (int *value)
{
    char buf[BUFFER_SIZE];
    readString (buf);
    *value = atoi (buf); /* Convert ASCII to integer data type. */
    return;
}

/* Read a floating-point value from the UART. */
void readFloat (float *value)
{
    char buf[BUFFER_SIZE];
    readString (buf);
    *value = atof (buf); /* Convert ASCII to FP data type. */
    return;
}
```

To display FP values, the basic idea is to split the FP value into its integer and fractional parts, and to write these parts out to a string as two separate integer values.

[4] ASCII stands for American Standard Code for Information Interchange, a character-encoding scheme originally based on the English alphabet. It encodes 128 specified characters into 7-bit binary integers, defining 95 printable characters, including the space, and 33 non-printing control characters such as `\n` and `\r`. Please see `https://en.wikipedia.org/wiki/ASCII` for more information.

```
1  /* Write a floating-point value to the UART. */
2  void writeFloat (float value)
3  {
4      char buf[BUFFER_SIZE];
5      double temp, integerPart, fractionalPart;
6
7      temp = modf (value, &integerPart);
8
9      /* Extract fractional portion to three digits of precision. */
10     temp = modf (1000 * temp, &fractionalPart);
11
12     sprintf (buf, "%d.%d", (int) integerPart, (int) fractionalPart);
13     writeString (buf);
14
15     return;
16 }
```

The `modf` C library function in Line 7 breaks the FP value into two parts: the integer part is stored in the pointer location passed as the second argument to the function, and the fractional value is returned by the function.

Use the functions in `uart_functions.c` to write the following simple functions:

- Use the UART to print out the string `Hello World` to the terminal on the host PC.
- Use the UART to print a sequence of integers from 1 to 100 to the terminal.
- Use the UART to print a sequence of floating-point values from 1 to 10 in increments of 0.1.

## Dynamic Control of LED Blinking Rate (10 Points)

Combine the above-discussed concepts to write a program that allows the user to dynamically control the blinking rate of the LED anywhere between 1 to 5 seconds. Begin with the LED blinking at 1 Hz and at run-time accept user input from the keyboard to dynamically change this rate between 1 and 5 Hz. Your code must check if the user input is within the desired range. If not, the LED must maintain the previous blinking rate.

The following code skeleton will help you get started.

```
1  #include <ti/devices/msp432p4xx/driverlib/driverlib.h>
2  #include <stdint.h>
3  #include <stdbool.h>
4  #include "uart_functions.h"
5
6  int main (void)
7  {
8      uint32_t i, j = 1;
9      uint32_t loopBound = 1000000; /* Bound set to default value. */
10     uint8_t ch;
11
12     WDT_A_hold (WDT_A_BASE); /* Stop watchdog timer */
13
14     initUART (); /* Initialize UART. */
15     writeString ("Established communication with the board");
16     /* Set P1.0, to output direction */
17     GPIO_setAsOutputPin (GPIO_PORT_P1, GPIO_PIN0);
18     /* Set DCO to 12 MHz. */
19     CS_setDCOCenteredFrequency (CS_DCO_FREQUENCY_12);
20     /* Main loop that runs on the board */
21     while (1) {
22         /* Peek into the ring buffer to see if we have a byte of data
23          waiting to be read. If so return 1, else return 0. */
24         if (peek ()) { /* Non-blocking function. */
25             ch  = readChar (); /* Read a character from ring buffer. */
26             j = ch - '0'; /* Convert ASCII to integer. */
27         }
28         /* Delay for some time using nops on the processor */
29         for (i = loopBound * j; i > 0; i --);
30         /* Toggle Pin. */
31         GPIO_toggleOutputOnPin (GPIO_PORT_P1, GPIO_PIN0);
32     }
33 }
```

Within the main processing loop, we continuously poll the ring buffer for any input data via the `peek` function in Line 24, which you will write. The `readChar` function, also to be completed by you, returns a byte from the ring buffer as an ASCII character. We convert this to an integer

value for further processing in Line 26.[5] Examining the ASCII table, we observe that if `ch` is the character returned, the corresponding integer value `j` can be obtained by simply subtracting the value of the ASCII character `'0'` from `ch`, that is `j = ch - '0'`. This integer value can be used to scale the delay loop to a value between 1 and 5 seconds. Note that the above code snippet does not contain the check to determine if the user input is within the desired range. You must add the additional logic. You may also have to experiment a little with the default value for `loopBound` to get the LED to initially toggle at 1 Hz.

The `initUART` and `writeString` functions can be found within the `uart_functions.c` file. You may add `peek` and `readChar` functions to that file. Feel free to add additional any additional functions that you deem to be appropriate to finish this task.

## Submission Instructions

Your C code must be clearly written, properly formatted, and well commented for full credit. A good synopsis of the classic book, *Elements of Programming Style* by Kernighan and Plauger, is available at `en.wikipedia.org/wiki/The_Elements_of_Programming_Style`. Another good reference is Rob Pike's notes on *Programming in C*, available at `www.maultech.com/chrislott/resources/cstyle/pikestyle.html`.

Once you have implemented all of the required features described in this document submit your code by doing the following:

- Run `Clean Project` to remove the executable and object files from the project folder. We must be able to build your project from source and we don't require your pre-compiled executables or intermediate object files. **If your code does not at the very least compile, you will receive a zero.**
- Zip up your project and upload the zip file using the Blackboard Learn submission link found on the course website.

[5] The ASCII table listing the value of each character is available online from `https://en.wikipedia.org/wiki/ASCII`.